AD-A015 125

DATACOMPUTER PROJECT

Computer Corporation of America

Prepared for:

Defense Advanced Research Projects Agency
Defense Supply Service

30 June 1975

279175

COMPUTER CORPORATION OF AMERICA

ADA015125

DATACOMPUTER PROJECT
SEMI-ANNUAL TECHNICAL REPORT

January 1, 1975 - June 30, 1975

Contract No. MDA903-74-C-0225
ARPA Order No. 2687

Submitted to:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139

DATACOMPUTER PROJECT
SEMI-ANNUAL TECHNICAL REPORT

January 1, 1975 to June 30, 1975

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by the U.S. Army Research Office, Defense Supply Service -- Washington under Contract No. MDA903-74-C-0225. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

TABLE OF CONTENTS

1 OVERVIEW

This report describes our work on the Datacomputer, a network data utility, from January 1, 1975 to June 30, 1975. The Datacomputer project is supported by the Information Processing Techniques Office of the Advanced Research Projects Agency of the Department of Defense. The current work is being carried out under contract number MDA903-74-C-0225. Related work discussed herein is supported by the Nuclear Monitoring Research Office of ARPA under contract number MDA903-74-0227.

Work during the reporting period has been concentrated on the development of Datacomputer Version 1, the first full service version of the Datacomputer. Previously, experimental service has been provided by various early versions of the Datacomputer, most recently (during the reporting period) by Version 0/11. Major bug fixes were the only programming done on Verion 0/11 during the reporting period. Preliminary goal setting and design for Version 2 of the Datacomputer consumed some small effort.

The work is described in detail in sections 2-8. Section 2 is a discussion of the Datacomputer architecture, with emphasis on the increasing levels of functional abstraction beginning with the hardware and moving outward. Section 3 is a report on the usage of the Datacomputer during the reporting period, and a discussion of new work being done in the user services and support area. Section 4 is a detailed discussion of the work on the Datacomputer software carried out during the period under discussion. Most of the effort was concentrated in this area. Section 5 discusses the on-going work of documenting the Datacomputer. Section 6 describes progress made in the area of Datacomputer hardware and operational support. Section 7 is a brief overview of the NMRO work and its implications for the Datacomputer in general. Finally, Section 8 is a catch-all for minor but important areas of Datacomputer development. The goals and concepts of the Datacomputer are discussed at length in Appendix 1, a paper delivered at the 1975 National Computer Conference.

2 System Description

The Datacomputer is a very large scale data storage facility with substantial data-management capabilities. Its design presupposes use as a data resource in a network of large-scale computers which are connected via medium speed (50,000 bits/second) communications lines. The data storage functions of the Datacomputer will support the storage of files as large as a trillion bits, and the hardware facilities will include a device with appropriate storage capacities (currently, an Ampex TBM is planned). The constraints of network bandwidth make the inclusion of powerful data selection and subsetting facilities imperative in the design of the data-management features of the Datacomputer. (To transmit one trillion bits at 50,000 bits/second requires approximately 231 days, assuming no hardware or software problems during the transmission.)

2.1 Levels of Functional Abstraction

Many large computer systems may be usefully examined in terms of their functional hierarchies. A level may be characterized in two ways. First, more fundamental operations which are provided by the previous level (and may already be abstractions themselves) are combined into new, more powerful, and more abstract operations. For example, the stream of magnetic flux reversals seen by the disk controller becomes a stream of fixed (or variable) length blocks of binary words when seen by the operating system. A subset of this arbitrary collection of unformatted words is presented to user programs as a "file" in a "file system". Second, intermediate functions exist to prevent certain combinations of operations which would damage system integrity from occurring, and to hide other functions entirely from the next level out.

The term normally used for the particular collection of functions available to any given level of a system hierarchy is "virtual machine". In many ways, the programmer working at level n in such a system may behave as if level n-1 were hardware; All n-1 functions are immutable and part of the machine environment. Using terms that will be explained in the rest of this section, the TENEX implementor programs a PDP-10; the SV programmer programs a TENEX (which looks a lot like a PDP-10 with some major abstractions); the Request Handler programmer programs an SV machine, and the ultimate user programs a Datacomputer. (We shall see that the set of functions presented by the Request Handler is equivalent to the Datacomputer virtual machine.)

The following four levels will be discussed in detail:

1) The hardware consists of a Digital Equipment Corporation (DEC) PDP-10 and its supporting peripher-

als, communications links to terminals and the ARPA Network, and a very large storage device.

2) The next level in the hierarchy, the TENEX operating system, is in direct control of the hardware resources and provides many services to the Datacomputer.

3) The programs known collectively as SV or Services are a pseudo-operating system which interacts with TENEX, managing input/output, scheduling, and storage strategies for Datacomputer files.

4) Finally, the Request Handler (RH) is the "user" level interface to the Datacomputer. It accepts control and data-management statements in "Datalanguage", and provides messages concerning the state of the Datacomputer job to the user. (Of course, data flows in both directions under the control of Datalanguage statements, and with the help of the other levels of the system.)

2.2 The Hardware Level

Conceptually, the hardware for a Datacomputer is quite simple. A processor of some sort is required along with some form of primary store (e.g., core). In addition, one needs a very large store (e.g., TBM) and a medium-to-high speed communications port. A great deal of efficiency can be gained by adding one or more levels of intermediate storage such as disc.

The hardware base of the Datacomputer as it is currently implemented consists of a processor, an address mapping device, three levels of store, medium and low speed communications lines, and several I/O devices.

The processor is a Digital Equipment Corporation PDP-10. CCA has a KA-10 CPU which is the oldest of several models of PDP-10 processor currently available. A Bolt Beranek and Newman "Pager" provides address translation for all memory references, and (along with software in TENEX) provides the illusion of a 256K (1K = 1024) word primary store regardless of the size of the physical memory.

The real primary store in the current Datacomputer is 208K words of 36 bit core memory. This includes five 16K DEC ME-10's and one 128K STOR-10 from Cambridge Memories, Inc. PDP-10 characters are typically stored five to a word, so this is the equivalent of slightly more than a million characters of memory.

The system has two types of secondary store. Five spindles of DEC RP02 disc (IBM 2314 equivalent) provide space for the TENEX file system. These hold about four million 36 bit words each, for a total of 20 million words. In addition, four spindles of CalComp 230 disc (IBM 3330 equivalent) are attached to the PDP-10 via a Systems Concepts SA-10 IBM data channel simulator. These discs each will store approximately 20 million words of data. Currently, they serve as the main file storage medium of the experimental Datacomputer service. Ultimately,they will serve as staging devices between the tertiary store and the PDP-10.

The real heart of the Datacomputer - the justification, in fact, for its existence - is the tertiary store. The tertiary store planned for the CCA datacomputer site is an Ampex Tera-Bit Memory (TBM). A version of the Datacomputer running at NASA/Ames utilizes a Precision Instruments 190 laser-based mass store (known as the Unicon). Thus, the Datacomputer can fairly be said to be free of dependence on any particular type of tertiary store. The set of such devices available today can be characterized as having very large storage capacities (as would be expected) and very high transfer rates (typically 6 megabits/second), but unfortunately, very slow access times (5-20 seconds). This set of characteristics mandates storage strategies which utilize very large data blocks on the tertiary storage device so as to minimize searching.

The Datacomputer's communications equipment consists of a connection to a Bolt Beranek and Newman Interface Message Processor which is in turn connected to the ARPA Network, plus a few low speed ports for the connection of local terminals to the PDP-10. The ARPANET connection is the Datacomputer's only channel to the outside world. All Datacomputer usage consists of messages and data passed back and forth through this port. Nodes in the ARPANET are connected by 50,000 bit/second phone lines, so the combined traffic of all concurrent Datacomputer users cannot exceed this transfer rate (except for the special case of usage from another host connected to the same IMP; see section 7.3). The low-speed lines are used by the system's developers and maintainers for communicating directly with TENEX. Since the Datacomputer never deals with these terminals directly, they will not be discussed in the remainder of this report, except in the section on hardware acquisition.

The input/output equipment on the CCA Datacomputer includes the usual array of peripherals - paper tape reader/punch, four DECtape drives (DECtape is a small, low capacity but extremely high reliability magnetic tape device used at CCA primarily for creating and re-loading the TENEX operating system), a line printer, and a 7-track, 800 bits-per-inch magnetic tape drive. The tape device is used for back up with the TENEX and Datacomputer disc systems to guard against the possibility of a catastrophic system failure causing the permanent loss of data.

2.3 The Primary Operating System - TENEX

The second level in the Datacomputer's functional hierarchy is the TENEX operating system. TENEX is a fairly sophisticated system written at Bolt, Beranek, and Newman for the PDP-10 beginning in 1969. Its intellectual predecessors include the DEC PDP-1 system designed at BBN, the Berkeley system for the SDS 940, CTSS and Multics from MIT, and the DEC 10/50 monitor for the PDP-10.

In the late 60's and early 70's, the PDP-10 seemed the most cost-effective system for small to medium scale scientific and research computing. It was especially popular in the ARPA community, and TENEX was conceived, at least in part, to meet the needs of that community. When CCA was looking for a machine on which to implement the Datacomputer concept, the TENEX, PDP-10 combination was quite attractive, and was therefore chosen as the Datacomputer base.

TENEX was designed primarily to support interactive time-sharing, with large LISP applications given special consideration by the system designers. In addition to providing traditional operating system functions such as scheduling and allocation of system resources, TENEX provides two separate but related abstractions which are especially useful in implementing systems such as the Datacomputer. They are: multiple virtual machines (limited to 256K words each by hardware addressing capabilites), and a powerful and flexible file system which includes the PDP-10's input/output devices as special files.

2.3.1 The TENEX Virtual Machine

A user program (such as the Datacomputer) running in TENEX may behave as if it were executing on a PDP-10 processor with 256K words of primary store and an extended instruction set which performs such high-level functions as file and memory management and communication with other processes. The extended instruction set may be utilized via the JSYS instruction, which passes control to the TENEX monitor. Normal hardware interrupts are not available to user processes, but simulated interrupts are provided by the system to facilitate I/O handling and inter-process communication.

A user's job may consist of several such processes (known in TENEX jargon as "forks") and the Datacomputer is in fact so structured. Processes of the same job, while dealt with separately by the TENEX scheduler, enjoy a special relationship with one another, and may intercommunicate much more freely than non-related processes.

Each process has an address space of 512 pages; each page contains 512 words. As in other virtual memory systems, only those pages actually being referenced by the process need actually reside in primary store. Those pages which are part of the process' address space but which are not currently being referenced are kept on secondary storage devices. The BBN Pager, mentioned in section 2.2, is responsible for the translation of virtual addresses (addresses in the process' virtual address space) into physical addresses in the primary store. When the word referenced by a virtual address is not found in primary store, its containing page must be brought in from the secondary store. The pager notifies the TENEX monitor, through a hardware fault, and the monitor (software) takes over the task of locating and providing the requested page to the process. First, a page of physical storage which has not been recently refrenced is located, and its current contents are saved if need be. Next, the newly needed page is read into the slot thus provided. Finally, all relevant page tables and secondary storage maps are brought up to date, and the TENEX scheduler is notified that the process which intially caused the page fault may be run again. This entire series of events takes place with no explicit help or knowledge from the process whose page is missing. Thus, no special programming steps need be taken by the designer of the user process, except for a general need to localize program and data references in order to minimize the (expensive) page faulting procedure just described.

2.3.2 The TENEX File System

The second TENEX provided abstraction of great usefulness to the Datacomputer is the TENEX file system, and in particular, its interaction with the TENEX virtual store described above. The file system of TENEX provides the user with a uniform view of devices connected to the PDP-10, and with a convenient approach to naming and accessing data stored on or transferred by those devices.

2.3.2.1 File Naming

The file system from a static viewpoint provides a multi-part name for each file, including the device the file is associated with, a directory name on the device, a unique identifier assigned by the user as the file's simple name, an extension identifying the intended use of the file, and an integer which indicates the version of the file. As files are opened by user processes, they are assigned small integers called Job File Numbers or JFNs with which future references are made.

Since the name of the device on which a file resides is part of the file's name as given by the user, no special provisions need be made by user processes for dealing with unusual devices. Of course, not all operations are applicable to all devices. (There's really no way to read data from a line printer.)

2.3.2.2 Byte-Oriented File Access

There are two distinct approachs to accessing files on TENEX. (Actually, this is an artifice. The TENEX monitor uses the same internal mechanism for both access modes.) The first view allows (requires) the user process to access the file as a stream of data bytes. The bytes vary in length from 1 to 36 bits. In general, access is sequential, but with some devices, it is possible to reset the current location of the "next byte" in the data stream. This access mode is the only one available for I/O devices such as terminals, printers, and paper tape handlers. With storage devices such as disks, tapes, and TBMs, it is simulated by the monitor which reads blocks of data from the device and returns bytes from the blocks as required.

2.3.2.3 File/Process Address Space Sharing

The second file access method, and the important one from the Datacomputer's viewpoint, views the file as an ordered collection of 512 word pages. Special JSYS routines in TENEX allow these pages to be mapped into arbitrary locations in the process's 512 slot address space. When this mapping is performed, the page of the file and the page of the virtual address space become indistinguishable; the two entities (file and process) are actually sharing the page, and changes made by the process to the page are immediately available in the file. (The file is strictly passive. It is an object, not an actor, and cannot modify its contents.)

Under rules carefully constructed to eliminate undesirable conflicts, several processes may map the same page into their address spaces. This results in sharing not only the logical page but also the physical memory in which it resides, thus ensuring that changes made to the page are immediately available to all processes which have it mapped in. It is possible to prevent sharing of file pages (on a per-file rather than a per-page basis) at file opening time, thus insuring the consistency of the data for the duration of a process' use of it. It is also possible to specify that a private copy of a page be made for any process which writes it, with no merging when the file is closed. This is especially useful during debugging and for programs which include modifiable data bases.

2.3.3 TENEX Modifications for the Datacomputer

The virtual machine provided by TENEX - a PDP-10 arithmetic processor with full memory capabilities and file/process address space integration - has proved to be quite hospitable to Datacomputer development. However, a few minor changes to the TENEX monitor have been necessary to optimize the Datacomputer's performance. First, routines to support the Calcomp 230 disk drives were added; and second, the scheduler was modified to give special considerations to the resource utilization patterns of the Datacomputer. Additional device handling code will be necessary when the TBM is integrated into the system.

2.4 The Pseudo Operating System - Services

The preceeding two levels of the Datacomputer system were not products of the development effort being discussed. They are described here because an understanding of their functions and capabilities is important to understanding the functions and capabilities of the two outer layers of the system - Services and the Request Handler.

These two levels constitute what could reasonably be called the "Datacomputer proper", and are the primary output of the Datacomputer project. They are conceptually and functionally separate - to the point of having separate staffs. This section discusses the Services programs (hereafter known interchangably, and in accordance with time-honored tradition, as SV).

SV functions as a pseudo operating system for the Datacomputer. It provides the basic functions of a traditional operating system in a form which is maximally convenient for the construction of a user-level Datacomputer interface (of which the current Request Handler is but one example). In particular, SV provides a specialized file system, stream oriented input/output facilities, and a set of scheduling/-monitor functions.

Access to SV functions for the Request Handler is via a special instruction known as "SVCALL". SVCALL's exist to manipulate the state of Datacomputer files including reading and writing pages from them; to perform input and output over the Datacomputer's ARPANET connections, and to handle special error conditions.

One useful side-effect of the Datacomputer's structure is in the area of transportability. While the fact that the Datacomputer is encoded entirely in Macro-10 assembly language militates against direct transportability, the existence of a clean, well-designed operating system interface (SV) means that very little re-design of user-level routines need be done. The logical structure can remain the same even though the routines must be re-coded into the language of the new host. Such an undertaking would still be an very large effort, but it would be considerably easier than starting from scratch.

2.4.1 The SV File System

The primary function of SV is to provide a convenient interface to the data storage facilities of the Datacomputer: the tertiary store and the staging device. A Datacomputer file as seen by the Request Handler programmer consists of an arbitrary number of "sections" (or sub-files), each of which is an ordered set of pages. (For convenience, SV pages are the same size as TENEX pages - 512 36 bit words.)

2.4.1.1 The Directory System

The Datacomputer file system may be thought of as a tree-structured hierarchy. At the top of the tree is a node whose conventional name is "%TOP". There are two types of nodes in the directory system - terminal and non-terminal. Terminal nodes (files) contain only data, and non-terminal nodes (directories) contain other nodes which exist at a lower level in the tree. Node creation is independent of the intended use of the node. In other words, a node in the tree is created, then at a later time it is specified whether it is terminal (a file) or non-terminal (a directory). Levels of the hierarchy are specified as a list of names connected by periods, such as "%TOP.DFTP.CCA". In the example, %TOP and DFTP are non-terminal nodes, and CCA may be either terminal or non-terminal (in the example, not enough context is present to determine which).

In addition to maintaining the directory hierarchy, the directory system provides protection for contents of nodes, whether other nodes or data. This protection takes the form of a set of "privilege tuples" associated with each node. A privilege tuple describes two things; the set of privileges allowed (or denied) to the user accessing the node, and the set of conditions which must hold before the node may be accessed at all (via this privilege tuple).

Just to give the flavor of privilege tuple application, one might specify that for a particular node, a user named "SMITH" may login to the node and create new nodes under it, but only if SMITH is connected to the Datacomputer from socket number 1000001 on ARPANET host number 31, and only if SMITH knows that the password assigned to the particular privilege tuple is "WASHINGTON". For a full discussion of privilege tuples, please refer to the latest Datalanguage manual.

This external view of the Datacomputer's file system - a tree-structured hierarchy with multiple protection classes enforced on each node in the tree - is dealt with transparently by the Request Handler. This means that the structure seen by, and the functions available to the ultimate Datacomputer user are essentially the same as those provided by Services to the Request Handler.

2.4.1.2 Access to Datacomputer Files

As mentioned above, a Datacomputer file is stored as an arbitrary number of sections, each of which is broken into 512 word blocks called pages. When the Request Handler wishes to access some page of a Datacomputer file, the following sequence of events must take place:

1) The file is opened. To open a file, RH supplies SV with the string representing the file's pathname in the Datacomputer file system (along with any needed passwords). SV determines that the current user is allowed to access the file in the manner requested (and the file exists), then returns a small integer, known as a Relative File Number or RFN. The RFN is the handle used by RH in all future references to the file until it is closed, at which time the RFN becomes invalid.

2) A buffer is allocated in the user process's address space. Buffers are managed by SV, but their allocation, freeing, and use is under the control of RH. A buffer is exactly the same size as a Datacomputer file page (and of a TENEX page). The buffer is identified by yet another small integer returned by SV.

3) If the page is being read (data already exists and is being referenced), an SVCALL known as PGRD is executed. This takes the RFN of the file, the section number, the page number within the section, and the buffer number into which the page is to be

read as inputs. After the call, the page is available in the buffer.

4) If the page is being created, data is first entered into the buffer by the Request Handler, then the page is written to the file by the SVCALL PGWR. Arguments are the same as with PGRD.

5) If the page is being modified, the sequence is PGRD, modify, PGWR.

6) When the Request Handler is through with the buffer and the file, the buffer is released by an explicit SVCALL, and the file is closed.

There are some problems with this file access strategy, primarily as a result of the fact that data pages lose their identity when in buffers. Since SV is perfectly happy to read the same page into two or more different buffers concurrently, it is possible for the Request Handler to unwittingly make two copies of a page; modify them both in the buffers; then write both back to the file, making the contents of the page at best uncertain. This case actually arose during the development of Datacomputer Version 1, and a non-trivial amount of time was spent locating and correcting the undesirable interaction.

2.4.2 The SV Input/Output System and Monitor

The input/output and monitor facilities provided by Services are fairly rudimentary when compared with the directory system. Input/output consists primarily of a set of connections to the ARPANET, with the ability to read and write buffers of data to/from a given connection. A special set of SVCALL's are provided for communication with the Datacomputer operator's console. The operator is consulted before particularly large requests are executed for user jobs, and certain kinds of messages about the state of the Datacomputer are routed there.

The Services monitor provides no particular facilities of its own, but is responsible for the creation/destruction of TENEX forks which represent particular Datacomputer sub-jobs. As users contact the Datacomputer via the network, they are assigned to a particular sub-job by the master process known as "Job Ø", which is just like any other Datacomputer process, except it has the monitor code enabled.

2.5 The User's Level - RH

The "outermost" level of the Datacomputer is known as the Request Handler. RH is in some sense an application program, since it is possible for a reasonably naive user to interact directly with it, via a specialized data-management language known as "Datalanguage". It would not be unreasonable to consider Datalanguage as the Datacomputer's order code.

Datalanguage and the Datacomputer were designed to be used by PROGRAMS running in other hosts on the network, and some of their characteristics which seem to contradict principles of good human engineering are a result of this assumption. Nonetheless, since all control interactions with the Datacomputer are expressed as strings of human-parsable ASCII characters, it is possible (and in fact the norm at CCA) for a human user sitting at a terminal which is capable of generating the ASCII control characters to interact directly and successfully with the Datacomputer. To avoid the anthropomorphization which usually creeps into descriptions of machine-machine interactions, this section is written as if the Datacomputer user were a real human being at a terminal. Tne reader is cautioned to bear in mind that this mode of use is decidedly secondary; that the Datacomputer is primarily a resource for machines and their programs.

2.5.1 User-Datacomputer Interactions

The Datacomputer maintains one or more input/output channels for the user. These are called "ports". All Datalanguage interactions flow over a particular port known as the "default port" or the "Datalanguage port". This port is the connection established when the user first contacts the

Datacomputer from the ARPANET. Data may flow over the default port or over auxiliary ports which are created by Datalanguage statements as the session progresses. It is preferable to use auxiliary ports for data for two reasons: first, only ASCII data may pass through the default port; and second, even though the data being passed is ASCII, great care must be taken to insure that it contains no characters which are treated specially when passed through the default port.

Datalanguage statements fall into two categories - commands and requests. In general, commands control the state of the user's Datacomputer process; open and close files, create nodes, modify privilege tuples, etc. Requests refer directly to the contents of files. A large part of Datalanguage is devoted to the detailed description of the contents of files, and the Request Handler makes extensive use of such descriptions in planning its actions.

2.5.2 Request Handler Structure

When the user first connects to the Datacomputer, Services initializes a new Datacomputer process, then passes control to the Request Handler. RH does some initialization of its own, then asks SV for the next line of input from the Datalanguage port. If the input line is a command, it is executed immediately. Requests are compiled, then executed.

There is no provision for storing requests in their compiled form. There are two reasons for this strategy: first, there is an assumption that two requests that are exactly alike are very rare indeed; second, the compilation time is trivial when compared with the execution time of requests on really large files.

2.5.2.1 The RH Compiler

The Request Handler's compiler is invoked for most requests. (A special subset of easy-to-handle requests are interpreted by a special module known as "slurp".) The compiler consists of three parts.

1) The first phase of compilation is handled by a routine known as the "pre-compiler". The pre-compiler takes the request as received from the user, does validity/syntax checking, and produces a new representation of the request known as "intermediate language". Intermediate language consists of a set of functions which are an abstract description of the entire set of operations which are legal on Datacomputer data. These functions are essentially the low-level machine language of the Datacomputer. They represent elementary operations such as "move an item from container 1 to container 2" with appropriate ancil-

lary information such as the type and location of containers 1 and 2. Most of the "smartness" of the Request Handler lies in the pre-compiler. It is completely responsible for the syntactic and semantic interpretation of user requests (but not their execution).

2) After the pre-compiler has abstracted and simplified the request, the intermediate language generated, and descriptions of the real files which are named in the request are fed to the rest of the compiler. This section is responsible for generating the instructions for actually moving data from one file (or port) to another under the control of the request. The output of this phase of the compiler is a data structure which contains all the messy loops, skips, and such for plowing through and pulling the data specified in the format requested from the file. The descriptions of each of these operations are called "tuples", although the exact referent of this term is somewhat ambiguous, as tuples are also the routines which interpret the data structures produced by the compiler.

3) Finally, the routines which actually execute the request on the data are, in some sense, part of

the compiler. Many of the tuples have distinct sub-routines which are responsible for their execution, and those routines constitute both the run-time environment and part of the compile-time data base of the compiler. Because of the multitude of data-types, byte sizes, etc. allowed by the Datacomputer, each tuple has many "modes", which are identified by bits in the data structure. For any given request, a particular set of modes is used, and a particular subset of the tuple code is executed. The last phase of the compiler walks through the tuple list that defines the request, and extracts the instructions which perform the tuple functions as constrained by the active mode bits in the tuples, producing the final "compiled request", which is executed with the real data.

2.5.2.2 Large File Considerations

For purposes of efficiency and implementation ease, files are sometimes broken down into smaller collections of data at various levels. Sub-groups at the only level currently in use are called "hunks", and are not used until file size approaches 20,000 records. In the future, more levels of sub-setting may be used internally.

3 Datacomputer Usage

During the reporting period, the experimental Datacomputer has been used by a quite diverse user population, with generally satisfactory results. The User Services group in the Datacomputer staff is responsible for interacting with such users, providing technical support, and maintaining various user-level programs which run the Datacomputer from remote network hosts.

3.1 Version 0/11 Usage Report

The 0/11 Datacomputer was the last experimental version of the system. It provided service to all Datacomputer users during the first half of calendar 1975. The following groups used significant amounts of Datacomputer time during the reporting period.

1) The Dynamic Modeling Group at MIT uses the Datacomputer for archival storage of their network host availability surveys. This system several times a day interrogates all ARPANET hosts, and notes whether they are currently serving remote users on the network.

2) A group at Harvard University used the Datacomputer for storing seldom-used files from their PDP-10 system. This use of the Datacomputer was

through a proram called "Datacomputer File Transfer Protocol", or DFTP for short.

3) A group at ETAC continues to plan for storing a very large data base of weather station reports on the Datacomputer when the TBM becomes available. Trial use of the Datacomputer has been made to test file formats and to attempt to increase understanding of the type of problems to expect when using such large data bases.

4) CCA uses the Datacomputer for archival file storage in preference to DECtape or large magnetic tapes.

5) The seismic data base application discussed in Section 7 has been making trial use of the Datacomputer. Like the weather people, most use so far has been to try out file formats, discover whether the Datacomputer has all the features needed to support the application, and generally gain familiarity witht the system.

6) Bolt Beranek and Newman is using the Datacomputer to store network-related information. The data is related to the IMP sub-network.

7) Several other network sites have moved towards making use of the Datacomputer as a large pseudo-on-line file storage facility, like Harvard is doing now. They include ISI, Sumex, BBN, Ames, Rutgers, and MIT (both the PDP-10 systems and Multics).

3.2 Planning for Version 1

The User Services staff planned, during the reporting period, for a smooth transition to Version 1 of the Datacomputer. The areas of particular concern included generating schedules of Datacomputer availability, producing a Version 1 Datacomputer reference card, and deciding how to handle user requests for information and aid.

There is potential conflict in the fact that a few users of the Datacomputer are quite large and quite important, while many others are relatively insignificant in terms of resources consumed and impact on system design. The problems occur because the smaller users often require just as much help with their applications as do the large and important users. As the Datacomputer moves towards becoming a service rather than an experimental facility, this problem must be addressed, and a plan for dealing with all sorts and sizes of users devised.

3.3 New User Contact

During the reporting period, many new potential users were informed of the Datacomputer's existence, and their problems were discussed in the context of a Datacomputer solution. Getting in touch with DoD people who might use/-need the Datacomputer is an important function of the User Services staff, and must be pursued with vigor in the future.

4 Software Development

As mentioned in section 1, most of the effort in Datacomputer development during the reporting period was concentrated on the release of Version 1 to ARPA Network users. Software development consumed most of the effort, and that work is presented in this section.

Software development can be broken down into three broad categories: SV development, RH development, and development of support programs running as separate TENEX jobs.

4.1 Services

For the Services group, the first half of the calendar year was primarily a time of feature enhancement; of fleshing out and solidifying what already exists. The SDAX mechanism was thought about in great detail and a large part of the code written. The directory cross-checker was thoroughly shaken down and debugged. Several minor features such as Temporary File Numbers (TFNs) were implemented (ususally as a result of a specific need of the RH group).

4.1.1 SDAX

The Special Disk Area Index design, known as SDAX, is designed to reduce the danger of undesirable interactions between multiple users of a single Datacomputer file. SDAX essentially takes the "file map", the mechanism which Services employs to remember where file data is actually located on physical storage, and breaks it out into a chain of maps which are searched when the current location of a data page is needed.

The goal of all this is to guarantee that the most up-to-date copy of a page in the file is always the one acquired, even though the same page may exist on TBM, on disk, and in primary store with all versions different.

SDAX is a major contribution to the Datacomputer's ability to serve as a central repository for large data bases which are to be updated and referenced by a variety of different users at different sites. Most of the work of providing this ability was complete by the end of the reporting period, although SDAX capabilities are not a feature of the Version 1 Datacomputer.

4.1.2 Directory Cross Checker

The Datacomputer, like any large, complex computer system, is subject to periodic interruptions due to the failure of computer hardware, software, and commercial power systems (not to mention operator error...). Whenever an interruption in operation occurs, it is possible that operations were in progress at the time of the interruption which had temporarily invalidated the integrity of information in the Datacomputer's directory system.

In the interest of recovering from such problems as gracefully as possible, the Datacomputer directories provide a substantial amount of redundant information about their structure. This redundancy costs something in storage space of course, but is crucial if faith in the Datacomputer as a safe repository of important data is to be justified. This faith is a necessary and important step on the road to a Datacomputer network service facility.

The directory cross-checker, which was completed during the reporting period, makes use of this redundant directory structure information to reconstruct damaged directories. It is typically run by Datacomputer operations personnel following a service interruption.

4.1.3 TFN's

During the development of the chaptered file software discussed in section 4.2.2, it became obvious that some facility for maintaining large scratch files in the Datacomputer was necessary. The mechanism chosen to satisfy this need is known as the Temporary File Number, or TFN.

A TFN is an SV entity which behaves just like a RFN when used in file-oriented operations. It is acquired by a special SVCALL, then used with no special precautions. The TFN is a much more efficient solution to the problem than the obvious alternative of creating and opening a dummy file, then deleting it at the end of the request (all of which could be done by the Request Handler), since The TFN mechanism allows short cuts to be taken in the management of SV's file status tables, as well as being easier for RH to deal with.

4.2 The Request Handler

The Version 1 request handler is based on that of Version 0/11. Re-writing from scratch was not thought necessary since that was done for major parts of the Request Handler during Version 0/11 dvelopment, providing an adequate base for Version 1. Nonetheless, almost all modules of the Request Handler were modified in some way as part of the development effort, and many new modules were written.

4.2.1 Restriction Removal/Cleanup

Since the Request Handler is the Datacomputer as seen by the Datacomputer user, it is especially important that this interface be as clean, straightforward, and transparent as possible within the constraints of the system's design and goals. With this in mind, a great deal of work went into removing the many special case restrictions which were in Version 0/11. These restrictions caused requests which seemed quite similar to the user to behave in completely different fashion, even to the point of not working at all. In addition, every attempt was made to assure that such restrictions did not creep into new code as it was being written.

The effort to maintain cleanliness and minimize restrictions was quite successful. In Version 1, requests which seem reasonable to the user will generally work as expected, and those which cannot be handled often return meaningful error messages. (Some improvement in error messages is definitely needed in the future.)

4.2.2 Chaptered Files and Updating

In Version 0/11 of the Datacomputer, one of the most glaring restrictions was the inability to modify the values of containers of variable length. This was especially galling in view of the fact that variable length conatiners are typically the best choice in terms of storage efficiency and ease of data handling where strings of characters (such as names, address) are involved. With these considerations in mind, the removal of such restrictions was given high priority in the development of the Version 1 Datacomputer.

Another feature targeted for inclusion in Version 1 was the maintainance of ordered files . the Datacomputer. The intention was to be able to delete and insert records at arbitrary locations within a file, and to maintain the original ordering of the file across such insertion and deletion operations.

Such ordering is typically maintained by use of one or more sort keys. A sort key is typically a named field within a record which takes on a set of unique values. An algorithm must be available which, given two values for the field will decide which comes before the other, or that they are identical. If it is necessary to discriminate between those records whose primary sort key fields are identical, then secondary keys can be set up, ad infinitum. (For example, we might want to maintain a file sorted by an individual's last name. If the last names are identical, then sorting is by first name, then by middle initial.)

These two somewhat separate problems - updating individual fields or containers within a record, and maintaining the logical ordering of the file at the record level - were seen as highly related and subject to joint solution. The solution that was ultimately adopted called for "chaptered files", and "an ordered CAT".

One major concern during the design was that locality of reference be maximized. If a new record is being added to a file, it is important that it reside as close as possible physically to the records with which it is associated. In particular, with respect to the TBM, it is crucial to minimize the number of blocks of data read. Strategies that result in fragmentation of related data encourage inefficient use of the TBM, and must be avoided.

Both the insert/delete and variable-length update issues can basically be seen as problems in free-storage management. Given an initial representation of the data in a file, space must be found for inserting new data, whether that data be an entire new record or an old record which no longer fits where it was initially allocated due to an increase in the size of one of its sub-containers.

Methods for managing such variable allocation problems are well known. Lists of space currently in use and of space available for use must be kept. Routines must be written to manage these lists. Extra space must be provided for at file creation time in order to allow for growth.

Of course, the file could be treated as a single, large free storage area with records allocated according to an algorithm which only takes into account the basic issues mentioned above. In the case of the Datacomputer, however, such an approach is not adequate because new or modified records would likely be allocated far from those to which they are most highly related, thus destroying the locality of reference which is so important when dealing with the TBM. (It should be noted that the relevant block size is quite large; approximately one million bits, or 28,000 words.) The solution we have adopted is to break the file into smaller units, which we call chapters. Each chapter is a complete free storage area in itself, and each contains records which are (hopefully) closely related.

When the free space in a chapter is exhausted, all subsequent calls for more space in it are directed to a specific overflow chapter. Thus, some loss of locality results, but it is kept to a minimum. (In practice, several original chapters share a common overflow chapter, and provisions for chaining additional overflow chapters together are made for the case in which the first overflow chapter is filled.)

A secondary goal of a chaptering scheme is to help in localizing searches based on the sort key. It was planned to maintain an index of chapters which would contain the range of values for the primary sort key which were found in each chapter. This would facilitate searches based on the value of the primary sort key, and would be a corollary to the Datalanguage inversions. This strategy is also known as the "ISAM Index" scheme.

4.2.2.1 The Container Address Table

As records are added, deleted, and moved within the file, its physical ordering moves further and further from its logical ordering. Of course, it would be possible to always sort the file when records are inserted, deleted, or modified into a new location. This approach, while fine for very small files, becomes physically impossible as file sizes approach a trillion bits. (Just to read a trillion bits into core at six million bits/second, the TBM's maximum transfer rate, requires almost two days.) We have therefore chosen to store the ordering information outside the data base in an auxiliary structure known as the Container Address Table, or CAT.

It is possible to have a CAT for any list in a Datalanguage file description. The CAT provides a quick access to list elements and is primarily a tool for increasing efficiency at run-time. For example, to obtain the nth element of a list of variable-length, delimited strings with no CAT would require reading through the first n-1 elements, searching for delimiters. If the same list had a CAT, obtaining the nth element would require only loading a pointer from the nth CAT slot.

The outermost list is treated specially, since each list element is in reality a record from the file. Chaptered files have CAT's automatically, but for non-chaptered files (also known as "pure base" files) the CAT option must be specified explicitly. If records contain varying length data and there is no CAT, then the file must be parsed by the Request Handler, record by record, until the desired record is obtained.

4.2.2.2 Problems with Original Design

The design specified above is basically a sound approach to the problems it attempts to solve. Difficulties, however, arose during the detailed implementation design. The basic difficulty arose from the fact that sort keys are not provided for by the Version 1 Datacomputer, and adding such a capability was beyond the scope of the chaptered file / updating effort. A generalized sort package is a desirable and scheduled feature for a future Datacomputer, but is not currently available.

The ordered file routines need some kind of handle on a new record in order to determine where in the file to insert it. This handle is normally the sort key, but with no sort keys, some other strategy is needed. Some argued for what became known as "magically ordered" files; the order to be determined as the order in which records are read into the datacomputer. Inserting in such a file would require exhaustively specifying the location into which the new record (or set of records) would go, then after the insertion the result would become the new "magic" order of the file.

This scheme seemed undesirable since there is no way for the system or the user to do validity checking on the files thus created. Of course, the user would typically have some sort key in mind when creating the file, but there would be no way for the Datacomputer to find out what that key was.

The decision finally made was not to implement ordered files in the Version 1 Datacomputer. Appending to and delcting from chaptered files will be possible, but no attempt at remembering the order of such appends/deletes is made. This scheme handles almost all users problems, and is considerably easier to implement than the original design.

Full variable length updating is, of course, provided in the final design for Version 1. The additional niceties of ordered files with full inserting and deleting will have to await the arrival of sorting facilities and sort key specification in the Datacomputer.

4.2.3 New Data Types

The Datacomputer's role as a central transfer point for data in a heterogenous network gives it the rather unusual requirement of being able to deal with almost any data type and machine representation of data. For example, strings can be represented in ASCII, EBCDIC, or BCD with various byte sizes. (Limitations in the TENEX network software restrict the byte sizes of data transmitted through ports to or from the ARPANET.)

In addition to representation of different data types, a set of conversions from one type to another must be available to that assignment, arighmetic, and comparison operations across data types are possible. Problems are sometimes encountered with such conversions. For example, the one's complement integer -0 (minus zero) has no representation in two's complement form. Liewise, it's not clear what collating sequence is appropriate when comparing ASCII and EBCDIC strings.

Almost all interesting data types and their conversions will be available in the Version 1 Datacomputer. The areas in which work is still needed are the many possible machine representations of floating point numbers and data elements with byte sizes greater than 36 bits.

4.2.4 List Command Improvements

The Datacomputer LIST command provides the user with information about files and directories on the Datacomputer. By specifying the appropriate options, the user can cause information about a single file or a group of files to be transmitted through the default port.

For example, the user might request a catalog of the names of nodes immediately contained in the directory to which he is logged in. Alternatively, the amount of storage assigned to a particular file might be requested.

During the reporting period, the LIST command was completely re-written. It now has much more powerful facilities for specifying file groups for output and a more complete set of options.

4.3 Support Programs

As previously discussed, the Datacomputer serves primarily as a resource for programs and software systems residing on various ARPANET hosts. With this primary goal, its design is not optimized for direct human use. A very large number of software interfaces to the Datacomputer will ultimately exist, running on a variety of host machines and with varying degrees of user awareness of the Datacomputer's existence.

In the short run, one program (subroutine) level and two user (terminal) level interfaces to the Datacomputer system are supported by CCA for use from TENEX hosts on the network. These programs allow users at TENEX sites to make elementary use of the Datacomputer's features.

During the period covered by this report, these programs were upgraded to work with the Version 1 Datacomputer. Features were added in some cases; in others, only compatibility changes were made.

4.3.1 DCSUBR

DCSUBR is a package of TENEX subroutines which user-level programs can call to manage their interactions with the Datacomputer (via the network,).

DCSUBR is used by the RDC program as well as various other small applications running on TENEX hosts around the network.

4.3.2 RDC

RDC is a program to Run the Datacomputer for a user at a terminal. Its basic function is to pass lines back and forth between the user and the Datacomputer, providing some (variable) screening of error/status messages on output, and standard type-in editing functions (such as character and line delete) on input.

RDC provides a very raw view of the Datacomputer. Probably, most real "users" will never need such a view, and RDC (and similar programs for other host systems) will exist primarily as a debugging and application development tool.

4.3.3 DFTP

The Datacomputer File Transfer Program, more popularly known as DFTP, provides the user with a simple off-line storage facility. This facility behaves much as if it were a large, slow tape drive with the capability of storing lots of bits and of maintaining a directory of file names available on the tape.

Files are stored on the Datacomputer (under DFTP) by explicit "put" operations, and must be retrieved by explicit "gets" before they are again available for use. This style of use is most appropriate for large, seldom-used files when disc space on the host is either scarce or expensive.

DFTP is a very elementary use of the Datacomputer, since none of the internal file structuring capabilities of Datalanguage are used. It is essentially a practical though trivial application of the Datacomputer.

5 Documentation

As the Datcomputer moves towards becoming a network service facility, the quality and the style of its documentation become increasingly important. First, end-user (external) documentation is especially crucial as the user population reaches the point beyond which individual hand-holding and documentation by phone call become impossible. Second, the stability and maintainability of any system dedicated to service require that its inner workings be carefully documented. Both internal and user documentation have been improved during the reporting period.

5.1 Version 1 Manual

Since Version 1 of the Datacomputer contains many new features, and since some old features have taken on new and different forms, the writing and publication of a Version 1 Datalanguage manual has been given high priority. Some material from the Version 0/11 manual has been retained, but most is completely new. The anticipated publication date for the new manual is early fall.

5.2 Support Program Documentation

At the end of the reporting period, documentation for the RDC and DFTP programs with Version 1 modifications was almost ready for distribution. This documentation is less formal than the Datalanguage manual, but is intended to be complete and accurate. DCSUBR documentation has been scheduled for the next reporting period.

5.3 Directory System PLM

The primary piece of internal documentation to emerge during the period is a program logic manual (PLM) for the Services directory system. This document discusses in detail both the internal structure of SV directories and the interface with the Request Handler. In addition, the data structures needed for communication between SV and RH are defined. The directory system PLM marks a large step forward in the documentation of the system internals.

6 Hardware / Site Progress

Activity in the hardware/site area of the Datacomputer effort was mostly confined to planning during the reporting period. The time thus spent should insure the smooth integration of the TBM system when it arrives.

It should be pointed out that the hardware and site work presented in this section is funded by the NMRO contract mentioned in the Overview rather than by the IPTO contract which is the primary subject of this report.

6.1 Site Improvements

Several changes in CCA's site will be necessary before the TBM can be installed. Among the most important are increasing the capacity of the machine room air-conditioning equipment, and re-arranging the machine room layout to make way for the new equipment. Several potential contractors were contacted about the site work, and extensive discussions were held with those who were interested.

6.2 TBM Negotiations

At the beginning of the reporting period, a contract was signed with Ampex Corporation for the delivery of CCA's TEra-Bit Memory system during August. At the end of the period, it appeared that problems with the TBM-PDP-10 interface specified in CCA's contract with Ampex might cause a delay in the delivery.

The interface normally sold by Ampex provides access to data on the TBM only via a disc staging device. CCA specified (and was promised by Ampex) an interface which provided for the reading of TBM data blocks directly into PDP-10 memory. This access style is necesary for the construction of the Datacomputer as originally designed. Decisions about the staging of data will be made by the Datacomputer after the data has been read into main memory.

6.3 TENEX Changes

During the reporting period, the 128K word memory system (a Cambridge Memories, Inc. system provided by Charles River Data Systems) was added to the CCA PDP-10 system. This addition was extremely important in that CCA was quite short of core space, but it was also the source of many problems during the period.

Since CCA was the first installation of the system, our machine was essentially the test bed for debugging a new memory controller design. The problems which arose from this situation, combined with quality control problems in the supposedly reliable core stacks caused many interruptions in TENEX/Datacomputer service. All problems were solved by the end of the reporting period, and service has become quite solid.

7 Seismic Data Base Support

As mentioned in the Overview, some work on the Datacomputer is funded under a separate contract from the Nuclear Monitoring Research Office of ARPA. A short discussion of it is included here because it is intimately related to the work of the primary Datacomputer development contract. In particular, the Ampex Tera-Bit Memory, about which so much has already been said, and the additional core mentioned earlier (which serves as a buffer for TBM blocks) are being paid for by the NMRO contract.

This work is specifically directed at establishing an on-line, real-time data base of seismic data from sites all over the US, and making that data available to analysts in a convenient way. The Datacomputer was chosen as the best vehicle currently available to do the job.

7.1 Overview

Since the system as envisioned will work in pseudo-real time, the ARPANET was chosen as the most appropriate communications medium available (as opposed to mailing tapes, high speed dial-up or leased lines, etc.). Seismic data is collected from sensors scattered all over the country, then transmitted to CCA over the network. At CCA, a small computer known as the Seismic Input Processor, or SIP, absorbs the incoming data (whose data rate is projected to be on the order of 20 thousand bits per second), and stores it on its own disc. At pre-determined intervals, the SIP connects to the Datacomputer (again via the network), and dumps the collected data into the Datacomputer at a very high rate. Thus, the Datacomputer-Seismic data colection center connection doesn't have to exist 24 hours a day, which reduces stress and strain on the Datacomputer most of the time. In addition, the SIP, being a mini-computer, is expected to have fewer failures than the Datacomputer.

7.2 SIP Acquisition

The SIP hardware, a PDP-11 with two spindles of 3330-equivalent disc and appropriate communications hardware, was delivered during the reporting period. Problems with the DEC discs, and with delivery of the cable which connects the SIP with the CCA TIP impeded progress, but by the end of the period, everything seemed to be running smoothly.

7.3 IMP/TIP Considerations

There has been some question as to whether the data rates envisioned for the seismic data application are in fact possible and practical with the current datacomputer hardware. In particular, the bandwidths expected through the CCA TIP are pushing the design limits of that device. Incoming data from the seismic data collection center is expected at 20,000 bits/second, and the SIP is expected to burst data to the Datacomputer at approximately 80,000 bits/second. As discussed above, the maximum bandwidth of an IMP or TIP where traffic to other network nodes is concerned is about 50,000 bits/second. When two hosts are connected to the same Network node, as is the case here with the SIP and the Datacomputer, that potential bandwidth is much higher.

The envisioned data rates are realizable, but with one problem. The TIP as a network node, due to the nature of its multi-function design, has less computing power available for the IMP sub-network functions than does a regular IMP. Therefore, some discussion was held during the reporting period as to the desirability of replacing the CCA Tip with an IMP.

The only problems with such a step are that CCA uses the TIP's special terminal handling characteristics extensively, so other arrangements for handling the devices currently attached to the TIP would be needed before the TIP could go away without adversely impacting CCA's work.

8 Other Activities

8.1 NCC Paper

A technical paper titled "The Datacomputer - A Network Data Utility" was prepared for the 1975 National Computer Conference. Authored by Thomas Marill and Dale Stern, it presented a conceptual overview of the Datacomputer system, and attempted to provide some flavor as to what styles of system usage were expected and planned for. It is included as Appendix 1 of this document.

8.2 Performance Monitoring

The performance of the Datacomputer has been a continuing interest of the staff. Several small, ad hoc tests have been run to get some feel for various aspects of Datacomputer performance. The primary result of these experiments has been to point up the desirability of more extensive, planned system metering and evaluation. With adequate instrumentation, we could begin to see what parts of the software would benefit most from tuning, what parts need complete re-design, what types of files are most expensive to handle, what data-management requests are least efficient. By the same token, we would be able to recomment to users file structures and access parameters which would be most efficient. Instrumentation and metering must be high priority items in future Datacomputer development.

8.3 Testing and Bug Monitoring

A substantial effort has been made in the area of Datacomputer reliability testing and monitoring. A set of standard test scripts was generated, and a continuing effort to construct tests for new and esoteric features was maintained. The goal of this effort was to have a set of benchmark procedures which would test all features of the Datacomputer in all interesting ways. This goal has not been reached, since new bugs appear fairly often which are not caught by the test procedures. By the time the Datacomputer becomes a standard network service, a complete set of tests should exist which could guarantee that new releases of the system would not contain any newly introduced bugs in things which used to work.

Reprinted from —
AFIPS — Conference Proceedings
Volume 44

The datacomputer—A network data utility*

by THOMAS MARILL and DALE STERN
Computer Corporation of America
Cambridge, Massachusetts

OVERVIEW

The Datacomputer is a large-scale data management and storage utility for use by a network of computers. The system is designed to provide facilities for data sharing among dissimilar machines, rapid access to large on-line files, storage economy through shared use of a trillion-bit store, and improved access control.

The present paper provides a conceptual overview of the system. Detailed treatment of the access language, software architecture, and relation to other developments in the database field[4,5,7,8,9] will be taken up in subsequent papers.

NETWORKS AND UTILITIES

Starting in the early 1960s, the idea that stand-alone computers could cooperate through communication facilities began to be explored,[1] and the concept of the resource-sharing network evolved.[2] In such a network, each computer draws on the others to supplement its own resources of hardware, software, and data. Today, the best-known network of this type is the Arpanet,[3] which ties together some forty-odd computers of different types.

Within a resource-sharing network, there is a natural tendency toward specialization of network nodes. Thus, for example, medium-scale machines with good time-sharing facilities will be used for interactive processes, but heavy scientific computation will tend to be passed to other machines that are particularly adept at such tasks. The factoring of problems into their constituents, the assignment of these constituents to the appropriate machines, and the recombination of results will tend to become an automatic process.

In the limit, specialized network nodes become what may be termed "utilities", that is, machines which perform a restricted range of functions solely for the benefit of the other machines. The Datacomputer is a network utility in this sense. It is entirely specialized for the performance of data management and storage functions. It offers resources to other machines on the net but does not draw on the resources of these machines.

One may speculate that the trend toward specialized network utilities will continue, and that the traditional stand-alone general-purpose machine will eventually disappear from the scene. The computer world envisioned in such a speculation might consist of a network containing a few very large Datacomputer-like systems, a few very large computational utilities ("number crunchers"), and a large number of small human-interaction units (such as intelligent terminals), having limited computational power and local storage. It is not clear that anything else is needed.

The justification of network utilities must primarily, of course, be made on economic grounds, by demonstrating that economies of scale and economies of specialization can be realized. In the case, specifically, of a data utility, there is an added justification: centralization reduces the severity of the technical problems of data sharing and may also alleviate some of the problems associated with privacy. If all data is kept in one box, one knows where to go look for it; by the same token, one knows where the control and protection procedures must be applied.

DESIGN CONCEPTS

Logically, the Datacomputer system can be viewed as a box which is shared by a variety of external processors, and which is accessed in a standard notation called "datalanguage." (See Figure 1.) The present section discusses the principal concepts underlying the design of the system.

Network data sharing

The Datacomputer provides data sharing services within a network environment. There are three principal design implications of this fact.

Data conversion

A database stored on the Datacomputer is sharable by all computers having access to the system. Thus, a single database is shared not only among users of different interests, but among users of different hardware. Character codes, floating point number representations, and word sizes vary from user to user; so do the representations of variable length and variable structure, as well as high level data structure attributes. The

* Work supported by the Advanced Research Projects Agency, Department of Defense.

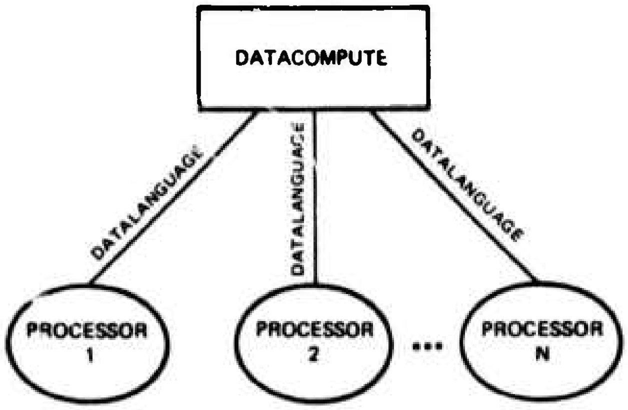

Figure 1—Logical view of datacomputer

Datacomputer system is required to perform translations between various hardware representations and data structuring concepts.

Characters, bytes, and numbers are stored under the control of the machine storing the data. The machine reading the data specifies the format it requires. As data is output, the indicated data conversions are performed.

Self-contained requests

In most approaches to data management (for example, the CODASYL approach[4]) the assumption is made that the data management system is in close contact with the application program. Thus the data management system can rely on the full capabilities of an application language (for example, COBOL) as being immediately available for processing the data.

This is not the case in a network environment, where the bandwidth between the application program and the data management system is relatively low. Thus, datalanguage must be designed to allow self-contained requests to be shipped to the Datacomputer to be executed there *in toto*.

Consider, for example, the problem of updating a large personnel file to reflect an across-the-board salary increase of 5 percent. In a conventional approach, the application program would sequentially obtain every record by making appropriate calls to the data management system, update the salary field, and replace the record (or build a new file) by calls to the data management system.

In a network, such an approach would be undesirable for large files, since it would require the entire file to be shipped twice, once to the application program, and once again back into storage. Accordingly, datalanguage is designed so that self-contained requests may be shipped to the Datacomputer from the application program. The Datacomputer itself performs the indicated function and signals the application program that the job has been completed, without requiring the records to be shipped to the application program.

Datalanguage does not, however, prevent the user program from generating a request which would cause the Datacomputer to ship an entire file to the requesting computer. That is, the Datacomputer can be used as a "file manager" in the style of the TABLON system,[5] as well as a data management system. For small files, this may be the preferred mode of use. For example, a short document that needs to be edited might best be shipped as a unit to the machine on which the editing will be performed, and then shipped back for storage.

Computer-oriented

The Datacomputer communicates with programs that run on remote machines. The fact of remoteness precludes the use of simple subroutine calls or similar means of communication conventionally used within a single machine. The communication, furthermore, is not with people at terminals, who can be expected to make intelligent responses when failures or unusual circumstances occur, but with programs. Hence, all synchronization messages, error messages, language statements, and file descriptions must be creatable and readable by programs; likewise, a facility for checkpointing by user programs is required.

Large on-line files

The Datacomputer is designed to have an on-line storage capacity of a trillion bits and to accommodate a wide variety of file sizes. In particular, the system handles files whose size approaches the total available space, that is, files in the trillion-bit range. To achieve efficient access to such files, two special facilities are included.

Inverted file structure

No adequate large file system can be designed without providing some mechanism for calculating the location of data in storage, given the attributes of the data to be retrieved. In the Datacomputer, this capability is achieved through a system of inverted files.*

At the user's option, files stored at the Datacomputer are totally or partially inverted. Once the file has been loaded, the inversion tables are maintained automatically by the system and need not be of concern to the user. Requests against a file may be composed without knowledge of the inversion options that have been selected for that file. The system will use the inversions, to the extent that they apply in a particular request, to limit the amount of sequential search that must be performed, thereby speeding up its retrieval process.

Multiple staging strategies

Internally to the Datacomputer, all data is physically organized into pages which move among the three levels of

* In a *direct file* one lists, for each entity, the properties of that entity. In an *inverted file* (also called *inversion*) one lists, for each property, the entities (or the location of the entities) having that property.

storage: primary (core), secondary (disk), and tertiary (mass store). The movement of pages is dictated by various staging strategies. The particular strategy used is selected by the system to optimize the requests currently being executed. The fact that the Datacomputer can itself select among the available strategies hinges on the fact that entire requests are transmitted to the system, informing the system at one time of the user's intent with respect to a given file.

Examples of staging strategies are as follows:

(i) Move the whole file to disk and work from disk. This strategy is applicable to small files that easily fit into the available secondary storage buffer area.

(ii) Move pages from tertiary store to core, process the pages, and output directly from core, bypassing disk. This strategy is applicable, for example, in the case where only a small portion of the data read from tertiary storage is to be sent to the user.

(iii) Break the request down so as to operate on segments of a file, and stage to disk one segment at a time. This strategy becomes particularly effective when information is available (from the inversion tables, for example) to indicate that some segments are not needed to fulfill the request, and can therefore be skipped.

Access regulation

The problem of controlling the access of programs to data in a general-purpose machine is notoriously difficult. By definition, a general-purpose environment allows the programs within it enormous latitude in the functions they can perform, and it appears that programs can often be written to circumvent existing access regulation procedures by taking advantage of coding errors in the operating system, hardware bugs, momentary malfunctions, or operational errors that arise in unexpected circumstances. Such hostile programs are sometimes able, without authority, to access data, delete data, or crash the system and prevent other users from legitimately accessing data.

In the environment of the Datacomputer, the situation is quite different, since the system is logically a closed, dedicated, special-purpose box, which responds only to a limited set of commands and does not provide a general-purpose computing facility. A hostile user program cannot be run on the box because the box does not run user programs. The approach can inherently provide stronger guarantees that programs without proper access authority will not be able to access or damage data contained in the Datacomputer. It is possible—though this needs to be explored further—that the Datacomputer approach lends itself to a proof that unauthorized access cannot occur.

Economy of scale and specialization

A variety of mass storage devices are coming on the market. These devices—the Ampex TBM, IBM 3850, Precision Instrument 190, among others—all have very

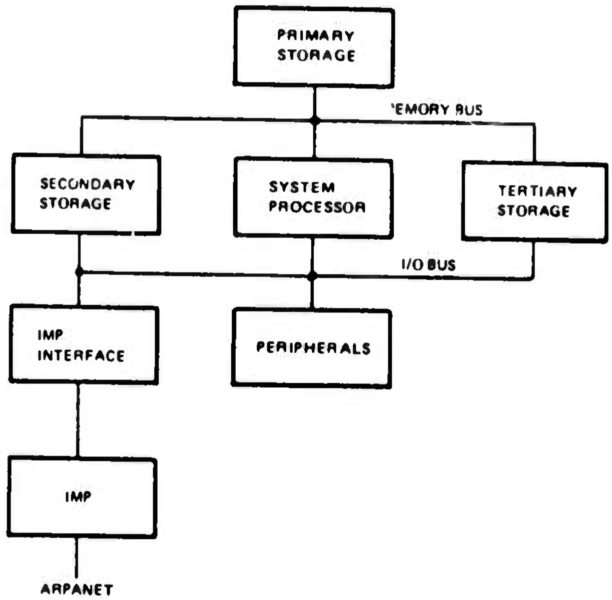

Figure 2—Hardware overview of system

high price tags, ranging from several hundred thousand to several million dollars, depending on configuration. They all, however, provide very low per-bit unit cost, with the lowest per-bit cost occurring in the largest configurations. Thus, while few stand-alone installations could afford the entry price, by pooling many users' requirements into a shared Datacomputer facility, the low per-bit cost of the mass store can be passed on to the users.

The savings can be substantial. Disk storage equipment (at the low end of the currently-available price-range) costs about $20 per megabit of storage. Mass stores cost about $1 per megabit, some twenty times less. All of these prices may be expected to decrease as technology improves, but there is no reason to suppose that the relative advantage of the economy of scale will not remain.

Certain additional economies can also be realized through specialization. In designing a specialized system it is possible to choose hardware and implement software in such a way as to optimize for the particular application, since there is no requirement to provide general-purpose services. In the particular case of the Datacomputer, it is possible to take advantage of new technologies as they become available, by making internal modifications and additions to the hardware and software of the system. This can always be done so long as datalanguage remains invariant, since the user program does not "see" the hardware or software of the system.

HARDWARE OVERVIEW

The architecture of the system is shown in Figure 2. The system processor is a DEC System-10 (PDP-10). Memory is present at three levels: core, disk, and TBM.[a] Peripherals are used for software development and for input

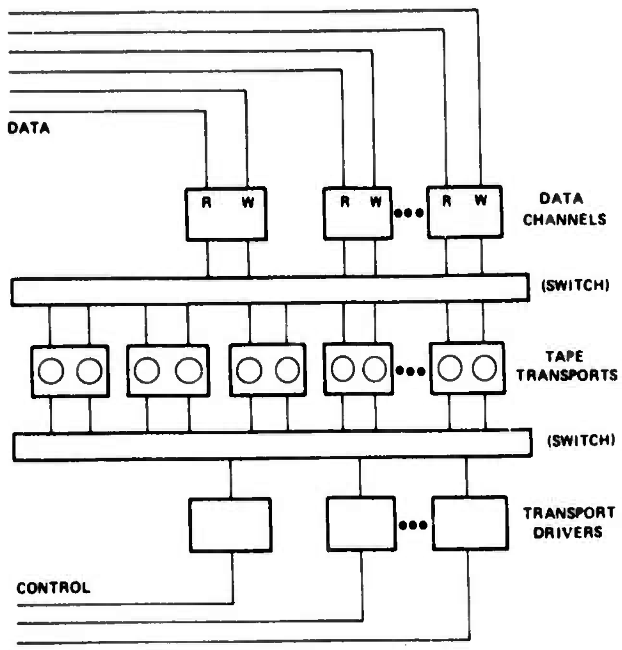

Figure 3—Ampex TBM configuration

of data from tape. The system is interfaced to the Arpanet IMP,[3] which in turn interfaces to two 50 kilobit/second telephone lines into the network.

Figure 3 shows in greater detail the configuration of the Ampex TBM tertiary storage subsystem. The system consists of three types of components, interconnected by two banks of switches. Channels are subdivided into two half-channels, one for reading and one for writing, each with a 6 megabit/second bandwidth. Each tape transport has two tapes, with a combined capacity of about 10^{11} bits; the maximum configuration has 64 tape transports. The transport drivers, or controllers, are switchable to any of the transports. In operation, a transport driver is switched to a tape transport, which is in turn switched to a data channel. Control information is passed to the transport driver, and data flows through the data channel. Data is written redundantly on the tape in a helical video scan with a density of 1 megabit/square inch. The average access time is 15 seconds.

DATALANGUAGE

Datalanguage is the language in which all requests to the Datacomputer are stated. Datalanguage includes facilities for data description, for database creation and maintenance, for selective retrieval of data, and for access to a variety of auxiliary facilities and services.

Datalanguage is a high-level language, which presents the user with a view of data which is independent of considerations of the physical devices on which the data is stored. The end user need not concern himself with search and scheduling techniques that are device-dependent.

Data representations are a special concern, because of the diversity of the user community. Data attrihutes which are ignored in other systems must be specified in this environment. The user must be able to map the data representations and data structuring concepts of his own machine onto those of the Datacomputer.

A basic characteristic of datalanguage is that all data is described. Descriptions are stored in the Datacomputer directory and are available to the user program in machine-readable format. A description contains the information needed to interpret the data, that is, information on data representations and structure.

An I/O transaction requires two descriptions: one for the data as it is stored—the "file description"—and one for the data as it comes in or goes out over the network—the "port description." Through the file description, the data administrator has control of how his data will be formatted on the Datacomputer. He can choose the representation that corresponds to the way the data will be accessed most frequently. In this way, the computation needed for reformatting is minimized, and higher bandwidths in and out can be achieved. Through the port description, the end user controls how the data as he sees it on his machine is formatted.

The data description facilities for ports and files are identical. In moving data between a file and a port, the Datacomputer performs the necessary reconfigurations of the data, including conversion from one elementary data type to another and pruning and reordering of branches in a hierarchical data structure.

Figures 4 and 5 show a port and file description, respectively, for a file of weather data. The port, called RESULTLIST, contains a list of "structs", called RESULT. Each RESULT has a city, date, and a minimum and maximum temperature. In this particular example, all of the data elements are fixed-length ASCII strings.

The file, called WEATHER, is tree-structured. Each of the 5,000 stations has some identifying information about the station and then a list of 31 weather observations. I=D indicates that the inversion option is being chosen for BSN, CITY, and REGION. This will cause the Datacomputer automatically to build inversion tables, which allow for content-hased retrieval without sequential search of the data base.

Figure 6 shows a retrieval request that selects and outputs data based on the value of REGION and the

```
CREATE RESULTLIST PORT LIST
       RESULT STRUCT, P=EOR
              CITY STR (22)
              DATE STR (3)
              TEMPERATURE STRUCT
                      MIN STR (4)
                      MAX STR (4)
                      END
              END
;
```

Figure 4—Sample datalanguage port description

maximum temperature.* The for-loop selects those stations with REGION equal to Massachusetts. Since the inversion option was chosen for REGION in the file description, the Datacomputer does not actually look at each station, but uses the inversion to find the selected stations. However, the user program submitting the retrieval request need not know that REGION is inverted; the request could be executed in any case.

For each of the selected stations, the second for-loop retrieves observations with TEMPERATURE.MAX greater than 300 (degrees Kelvin). Transmittal of data is indicated by assignment. Each RESULT record has four values: CITY, DATE, TEMPERATURE.MAX, and TEMPERATURE.MIN.

This example maps data from a 2 level tree-structured file to a 1-level tree-structured port. The observations in the port, unlike the ones in the file, are not organized by station; rather, the CITY is repeated for each output record.

In order for a request submitted by another machine to be executed, the Datacomputer must synchronize with external processes. Figure 7 shows the same request as above, along with the messages needed for synchronization of the Datacomputer and the other process. The first five characters are coded to be machine-processable. For example, the .I200 message indicates that the Datacomputer is ready for more datalanguage. Other messages direct the user program to send data, to send a new request, to close out the transactions, etc.

```
CREATE WEATHER FILE LIST(0,5000), P=EOF
  STATION STRUCT
      BSN         STR(6), I=D
      CITY        STR(22), I=D
      REGION      STR(22), I=D
      WORLD       STR(22)
      OBS         LIST (31)
       OBSERVATION       STRUCT
        DATE    STR(3)
        TEMPERATURE      STRUCT
           MIN  STR(4)
           MAX  STR(4)   END
        PRECIP  STR(4)
        WINDS   STRUCT
           SPEED         STR(4)
           GUSTS         STR(4)
           DIRECTION     STR(4)   END
        VISIBILITY       STR(4)
        CLOUDS  STR(4)
        GENERAL STR(4)
        PRESSURE         STR(4)   END
 END ;
```

Figure 5—Sample datalanguage file description

* The symbols "/*" and "*/" are delimiters for comments.

```
OPEN RESULTLIST ;
OPEN WEATHER ;

FOR WEATHER.STATION WITH REGION EQ 'MASSACHUSETTS
   FOR RESULTLIST.RESULT, OBSERVATION WITH TEMPERATURE.MAX GT ' 300'

      /* 300 KELVIN IS 80 FAHRENHEIT, THAT IS HOT
         IN OCTOBER IN MASSACHUSETTS */

        RESULT.CITY = STATION.CITY ;
        RESULT.DATE = OBSERVATION.DATE ;
        RESULT.TEMPERATURE = OBSERVATION.TEMPERATURE ;
        END ;
    END ;
```

Figure 6—Sample datalanguage retrieval request

STATUS OF DEVELOPMENT

The Datacomputer has been offering service on the Arpanet since late 1973, using disk-storage only. Installation of the TBM tertiary store is scheduled for 1975.

The system is undergoing a phased development; successive versions offer increased capabilities to users by providing increasingly larger subsets of datalanguage. Thus, the development proceeds in an operational setting in which design errors and implementation bugs can be discovered early through feedback from actual users.

The version of the system currently offering service on the Arpanet (Version 0/11) is an intermediate version which provides adequate facilities for many applications, such as the ones described below, but by no means for all applications. An enhanced version is scheduled for mid-'75, with additional capabilities planned beyond that date.

As successive versions extend the range of datalanguage, previously written user programs incorporating datalanguage can either remain invariant or may require small modifications. Changes to Datacomputer hardware, such as the installation of TBM, are not reflected in datalanguage, and therefore require no change to user programs.

APPLICATIONS

In this section three representative applications of the Datacomputer are discussed. The first two are in operation, and the third is currently being developed.

On-line information retrieval

As a service to the Arpanet community, a program at MIT Project MAC automatically surveys the status of all Arpanet hosts three times per hour around the clock. At each run, the SURVEY program attempts to connect up to each host, and stores the data, time, status, and response time. The data is automatically passed to the Datacomputer, where the historical SURVEY file is then updated by the current data.

As a companion to the data-collection facility, SURVEY provides on-line user functions that allow the database to be interrogated. A user on the network logs into MIT and composes his request for information in the on-line language supplied as part of the SURVEY application. The SURVEY program translates these requests into datalan-

```
;J200 11-11-74 1207:53  RHRUN: READY FOR REQUEST
.I210 11-11-74 1207:53  LAGC: READING NEW DL BUFFER
 OPEN RESULTLIST ;
;U000 11-11-74 1208:09  DHKD: ADDING PUNCTUATION
;J209 11-11-74 1208:09  RHRUN: EXECUTION COMPLETE
;J200 11-11-74 1208:09  RHRUN: READY FOR REQUEST
.I210 11-11-74 1208:09  LAGC: READING NEW DL BUFFER
 OPEN WEATHER ;
;J209 11-11-74 1208:12  RHRUN: EXECUTION COMPLETE
;J200 11-11-74 1208:12  RHRUN: READY FOR REQUEST
.I210 11-11-74 1208:12  LAGC: READING NEW DL BUFFER

.I210 11-11-74 1208:12  LAGC: READING NEW DL BUFFER
 FOR WEATHER.STATION WITH REGION EQ 'MASSACHUSETTS
.I210 11-11-74 1208:14  LAGC: READING NEW DL BUFFER
    FOR RESULTLIST.RESULT, OBSERVATION WITH TEMPERATURE.MAX GT ' 300'
.I210 11-11-74 1208:14  LAGC: READING NEW DL BUFFER

.I210 11-11-74 1208:14  LAGC: READING NEW DL BUFFER
        /* 300 KELVIN IS 80 FAHRENHEIT. THAT IS HOT
.I210 11-11-74 1208:15  LAGC: READING NEW DL BUFFER
          IN OCTOBER IN MASSACHUSETTS */
.I210 11-11-74 1208:15  LAGC: READING NEW DL BUFFER

.I210 11-11-74 1208:16  LAGC: READING NEW DL BUFFER
          RESULT.CITY = STATION.CITY ;
.I210 11-11-74 1208:18  LAGC: READING NEW DL BUFFER
          RESULT.DATE = OBSERVATION.DATE ;
.I210 11-11-74 1208:18  LAGC: READING NEW DL BUFFER
          RESULT.TEMPERATURE = OBSERVATION.TEMPERATURE ;
.I210 11-11-74 1208:19  LAGC: READING NEW DL BUFFER
          END ;
.I210 11-11-74 1208:20  LAGC: READING NEW DL BUFFER
     END ;
;J205 11-11-74 1208:23  RHRUN: SUCCESSFUL COMPILATION
.I241 11-11-74 1208:26  OCPOO: (DEFAULT) OUTPUT PORT OPENED
SOUTH WEYMOUTH         283 281 320
SOUTH WEYMOUTH         287 279 320
NORWOOD                288 271 326
.I261 11-11-74 1208:29  OCPOC: (DEFAULT) OUTPUT PORT CLOSED
;J209 11-11-74 1208:30  RHRUN: EXECUTION COMPLETE
;J200 11-11-74 1208:30  RHRUN: READY FOR REQUEST
.I210 11-11-74 1208:31  LAGC: READING NEW DL BUFFER
```

Figure 7—Sample datacomputer output and protocol messages

guage, sends the datalanguage to the Datacomputer, receives output from the Datacomputer, and presents the output to the user at his on-line terminal. The database management functions are all performed at the Datacomputer.

File management

A university computer center on the Arpanet routinely uses the Datacomputer system in a file management application, by means of a program called Datacomputer File Transfer Program (DFTP), which runs at the computer center. This program allows a local user program to store a file on the Datacomputer, retrieve a file, and add and delete a directory node. All DFTP-Datacomputer dialogue (datalanguage and protocol messages) is invisible to the user; the operation is automatic; access control mechanisms are provided. DFTP is particularly useful in this situation because the computer center is short of online storage for its users, and alternative solutions would involve magnetic tape and manual intervention.

Large shared file with multi-host access

In an application under development, the Datacomputer will be used as a central storage location and distribution point for a large database of seismic data collected from around the world in real time. Data will flow through the Arpanet to the Datacomputer, where it will be stored on-line. The data rate into the Datacomputer will grow over time, reaching a maximum of about 20 kilobits per second, 24 hours per day (6.3×10^{11} bits/year). Users of the data will be able to access the central database from any host machine in the Arpanet. By sending proper datalanguage requests to the Datacomputer, the host machine will be able to select arbitrary subsets of the large file and have these subsets shipped back in formats suitable for the particular host.

ACKNOWLEDGMENTS

Many people in the Datacomputer group at CCA have contributed to the development of the system, and their contribution is gratefully acknowledged. Special thanks are due to Richard A. Winter, Hallam G. Murray, David W. Shipman and Jeffrey M. Hill.

REFERENCES

1. Marill, T. and L. G. Roberts, "Toward a Cooperative Network of Time-Shared Computers," *Proceedings AFIPS Fall Joint Computer Conference*, 1966, pp. 425-431.
2. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *Proceedings AFIPS Spring Joint Computer Conference*, 1970, pp. 543-549.
3. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *Proceedings AFIPS Spring Joint Computer Conference*, 1970, pp. 551-567.
4. *CODASYL—Data Base Task Group Report*, ACM, New York, October 1969 and April 1971.
5. Gentile, R. B. and J. R. Lucas, "The TABLON Mass Storage Network," *Proceedings AFIPS Spring Joint Computer Conference*, 1971, pp. 345-356.
6. Damron, S., J. R. Lucas, J. Miller, E. Salbu, and M. Wildman, "A Random Access Terabit Magnetic Memory," *Proceedings AFIPS Fall Joint Computer Conference*, 1968, pp. 1381-1387.
7. Codd, E. F., "Recent Investigations in Relational Data Base Systems," *IBM RJ*, 1385, April 1974.
8. *Model 204 Database Management Software System-User Language Reference Manual*, Computer Corporation of America, September 1974.
9. Canaday, R. H., R. D. Harrison, L. L. Ivie, J. L. Ryder, L. A. Wehr, "A Back-End Computer for Data Base Management," *Communications ACM*, 1974, pp. 575-582.

61